6-1-1997

Parallel Disassembly by Onion Peeling

Shiang-Fong Chen
Iowa State University

Shuo-Yan Chou
National Taiwan Institute of Technology

Lin-Lin Chen
National Taiwan Institute of Technology

James H. Oliver
Iowa State University, oliver@iastate.edu

Parallel Disassembly by Onion Peeling

Abstract
For some assembly structures, parallel disassembly of components is necessary in order to reach a particular internal component. Due to the large number of possible combinations, the parallel disassembly problem is not easily solved in a general form. In order to reduce the time complexity of finding a disassembly sequence, this paper introduces a simplified mating graph and develops a data structure to facilitate an efficient parallel disassembly algorithm. This algorithm takes Max {O(N3), O(E)} time to find an efficient sequence to reach a particular component, where N is the number of components and E is the number of mating faces. Separability testing is incorporated to determine whether the query component can be disassembled and moved to infinity without obstruction.

Keywords
Manufacturing, Algorithms, Testin

Disciplines
Manufacturing

Comments
This article is from *Journal of Mechanical Design* 119 (1997): 267–274, doi:10.1115/1.2826246. Posted with permission.

Shiang-Fong Chen

J. H. Oliver

Department of Mechanical Engineering,
Iowa Center for Emerging Manufacturing Technology,
Iowa State University, Ames, IA

Shuo-Yan Chou
Department of Industrial Management,

Lin-Lin Chen

Graduate School of Engineering Technology,
National Taiwan Institute of Technology,
Taipei, Taiwan

Parallel Disassembly by Onion Peeling

For some assembly structures, parallel disassembly of components is necessary in order to reach a particular internal component. Due to the large number of possible combinations, the parallel disassembly problem is not easily solved in a general form. In order to reduce the time complexity of finding a disassembly sequence, this paper introduces a simplified mating graph and develops a data structure to facilitate an efficient parallel disassembly algorithm. This algorithm takes Max {$O(N^3)$, $O(E)$} time to find an efficient sequence to reach a particular component, where N is the number of components and E is the number of mating faces. Separability testing is incorporated to determine whether the query component can be disassembled and moved to infinity without obstruction.

1 Introduction

Motion problems are studied in many fields such as robotics, CAD/CAM systems, computer graphics and assembly process planning. The basic task is to find a path or sequence of motions to move a component to a specific position without collision. If several mechanistic factors such as force and friction are ignored, motion planning becomes a purely geometrical problem.

There are two major motivations for the study of disassembly problems. One is for generating assembly procedures, and the other is for maintenance. It is often more feasible to analyze the assembly problem from the view of disassembly. On the other hand, if a component inside a structure is broken, a disassembly procedure is required to replace the component. Generally, disassembly problems deal with components having adjacent faces. In the following discussion, the term *query component* refers the component or subassembly whose accessibility is currently under consideration.

The Disassembly Tree (DT) (Woo and Dutta, 1991) is a data structure for representing a disassembly sequence. One disassembly sequence can be determined by traversing a path from top to one leaf of the DT, and the nodes are the components to be removed. Woo and Dutta classified assemblies into two groups based on their logical sequence. One is *partially ordered assembly,* the other is *totally ordered assembly.* If every node in the DT has only one parent, it is a *totally ordered assembly,* otherwise it is a *partially ordered assembly.* In partially ordered assembly, if disassembling a component requires clearing it with other components in parallel, it is a *parallel assembly.* If disassembling a component requires clearing other components sequentially, it is a *sequential assembly.* In the disassembly process, if disassembling the query component requires only one translation, it is called a *one-translation assembly.* If disassembling the query component requires changing the moving direction, it is a *multitranslation assembly.* These assembly classifications and their corresponding DT's are shown in Fig. 1.1.

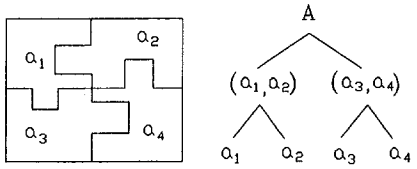
If adjacent faces can be separated totally, the query component is considered to be disassemblable, but interference of the query component with other objects during translation is generally not considered. On the other hand, most separability problems deal with components having no adjacent faces (Sack and Toussaint, 1987; Dehne and Sack, 1987; Toussaint, 1989) and only if the query component can be translated to infinity is it considered to be separable. If P and Q are two simple polygons with n and m ($n > m$) vertices respectively, their movability (separability) wedge can be computed in $O(n \log m)$ time (Sack and Toussaint, 1987). The movability wedge of P with respect to Q is the set of all directions in which P can be translated to infinity without collision. In this paper the separability algorithm, by Sack and Toussaint (1987), is referred to as the *separability algorithm.*

Actually, if a component is disassemblable, it is not necessarily separable. Fig. 1.2 shows that component a is disassemblable, but after the adjacent faces are totally disjoint, component a is determined to be inseparable. In this paper assemblies having adjacent faces are considered. If the query component is separable after being disassembled, it is referred to as *removable,* so the disassembly trees constructed in this paper are actually *Removability Trees* (RT).

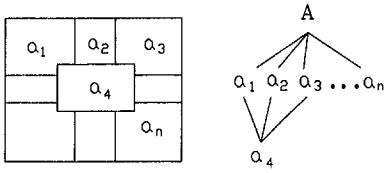
Parallel Disassembly. Since parallel disassembly addresses clearing k ($k > 1$) components in parallel, nodes in the DT may contain more than one component. Dutta and Woo (1992) present a parallel disassembly algorithm which is employed only when no single boundary component can be removed. Actually, to save disassembly time, totally ordered assemblies and sequential assemblies also can be disassembled by a parallel procedure. In this paper, parallel disassembly is employed for both totally and partially order assemblies.

If structures are disassembled by a parallel procedure, the problem becomes how to decide which components can be removed in parallel. Some algorithms address this problem by merging the inseparable components together (De Floriani and Nagy, 1989) or merging the internal components with the boundary components (Dutta and Woo, 1992). However, if a structure is very complex and contains many components, it is computational expensive to decide which components can be merged together because there exists an exponential number of combinations for merging. Another approach solves the parallel disassembly problem by computing a "weight" between adjacent components and merging component pairs which have high weights. The weights are proportional to the difficulty of their corresponding operations (Homen de Mello and Sanderson, 1990; Lee and Shin, 1990) and are often assigned based on user's experience and judgment. Lee and Wang (1993) solve the disassembly problem by physical reasoning on interconnection forces, with complexity $O(N^4 + N^4 2^{N/e})$, where $e = N/M$, N is

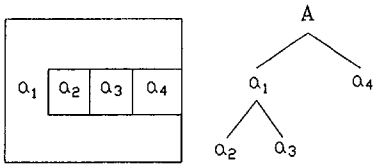
Contributed by the Design Automation Committee for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received Aug. 1994; revised Aug. 1996. Associate Technical Editor: D. A. Hoeltzel.



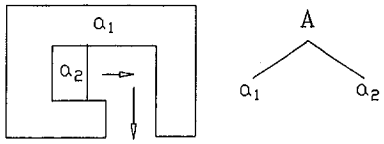
(a) Partially Ordered and Parallel Assembly



(b) Partially Ordered and Sequential Assembly



(c) Totally Ordered



(d) Totally Ordered and Multi-Translation

Fig. 1.1 Classification of assemblies

the number of components, and M is the number of nodes after the merging process.

In order to avoid the intractable nature of the general parallel disassembly problem, this paper devises a very efficient and simple algorithm by merging *all* internal components together and simplifying the mating graph into a succinct form. The main idea is to recursively reduce the original complex problem into several simpler ones rather than solving it directly. Furthermore, this paper analyzes the time complexity for finding all disassembly combinations from the view of computational geometry. The disassemblability of the query component is addressed first. If the query component is disassemblable, its separability is tested to determine whether it can be moved to infinity. Only components which contact with the outside are assumed to be removable. This is not an unreasonable assumption because an object must be exposed to be accessible. Finally, removal motion is restricted to 1-translation.

The remaining sections are organized as follows. In Section 2, in order to implement the parallel disassembly algorithm, a simple procedure is developed for testing the monotonicity of mating faces to determine disassemblability, and the mating graphs with or without cycles are also discussed. In Section 3, an algorithm for finding an efficient path to reach a query component is described. The implementation of the parallel disassembly algorithm and the conclusion are presented in Section 4.

2 Monotone Chain in an Assembly

The subassembly remaining after the query component has been removed is referred to as the *main-subassembly* and the faces shared by the query component and the main-subassembly as *mating faces.* It has been shown that considering the mating faces is sufficient for checking their disassemblability (Woo and Dutta, 1991). Since only 2D cases are considered here, a sequence of mating faces form a mating chain of polygon edges.

If the mating chain is a monotone chain, the query component is disassemblable, otherwise it is not. The monotonicity of a mating chain is tested by considering the unit normals of the mating faces on the unit circle. If the span of all unit normals is within a semicircle, the query component is disassemblable, otherwise it is not disassemblable. By observation it is easy to distinguish whether the span of unit normals is within a semicircle or not, but to implement the algorithm, a specific procedure is required to characterize the relationship. The following algorithm checks the monotonicity of a mating chain in linear time.

Semicircle. Input: A mating chain with k vertices, v_1, v_2, . . . , v_k, in that order.

BEGIN

(1) For each edge v_iv_{i+1} of the mating chain, find the unit normal u_i that points to the left of the edge.

(2) Divide the $k - 1$ unit normals u_i, according to the cross product of u_1 and u_i, into two sets U^+ and U^-, where

$$U^+ = \{u_i: u_1 \otimes u_i \geq 0\}$$

$$U^- = \{u_i: u_1 \otimes u_i < 0\}$$

(3) For each unit normal u_i in U^+, find the angle θ_i between u_1 and u_i. Let $\theta^+ = \max\{\theta_i: u_i \in U^+\}$. Similarly, for each unit normal u_i in U^-, find the angle θ_i between u_1 and u_i. Let $\theta^- = \max\{\theta_i: u_i \in U^-\}$.

(4) If $(\theta^+ + \theta^-) > 180°$ then the mating chain is not monotonic. Else, the mating chain is monotonic. □

END

The function *Semicircle* clearly requires only $O(k)$ time to determine if a mating chain with k vertices is monotonic or not. Fig. 2.1 illustrates this algorithm.

The mating graph is constructed by removing the boundary of the assembly, and it may be cyclic or acyclic. Figure 2.2 shows an assembly and its corresponding mating graph. In this section a mating graph traversal algorithm is developed to determine which components are removable.

In an acyclic graph, there is only one route from one vertex to another vertex, i.e. there are no loops in the graph. This property will be applied in the parallel disassembly algorithm later. Here, the vertices in a mating graph are classified by their

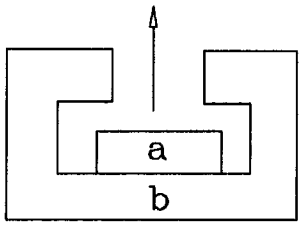


Fig. 1.2 Component a is disassemblable but not separable

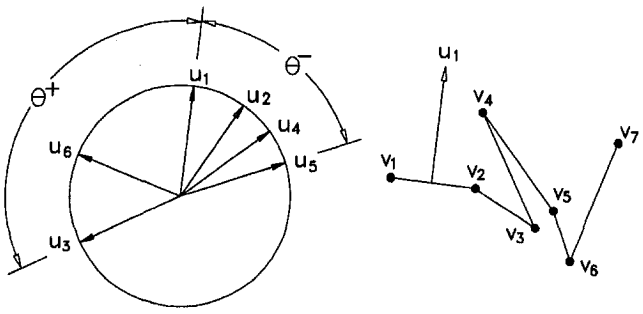


Fig. 2.1 Illustration of the function semicircle

degrees. The degree is the number of edges that intersect at a vertex. A *terminal vertex* (v_T) is a vertex having degree one. An *intersection vertex* (v_I) is a vertex having degree greater than two. A vertex having degree two is called a *simple vertex* (v_S). An *edge-list* in a mating graph is a sequence of edges with v_I or v_T as end points, and all the vertices between the two end points are *simple vertices*. Figure 2.2(*b*) shows examples of v_T, v_I, v_S, and an *edge-list* in a mating graph.

Lemma 2.1 If every component in an assembly is exposed to the outside, and there is no empty space inside the assembly, the mating graph of the assembly does not contain a cycle.

[*Proof*] Suppose to the contrary there exists a cycle in the mating graph, since the assembly does not have empty space inside, all the components enclosed by the cycle are not exposed to the outside. This contradicts the fact that all the components in the assembly are exposed to the outside. Therefore, there cannot exist any cycle in such a mating graph. □

If a mating graph has no cycle, it may be a tree or a forest. Because the same properties exist in each tree of a forest, the case of a forest is not discussed further. Assuming the mating graph is a tree, the following lemmas are established.

Lemma 2.2 The number of components is equal to the number of terminal vertices.

[*Proof*] Every terminal vertex is shared by two components, and every component can be defined by two terminal vertices, so the number of components is equal to the number of terminal vertices. □

Lemma 2.3 The number of intersection vertices is at most (N − 2), where N is the number of components.

[*Proof*] This Lemma is proved by construction. First, a path P along the mating graph between an arbitrary pair of v_T is traced out and the two v_T's are marked. Since the mating graph is a tree, such a path is unique and non-self-intersecting. P divides the region bounded by the boundary of the assembly into two domains. Since the mating graph is a tree, all the edges in the mating graph are connected. If there are more than two components in the assembly, there must be at least one v_I on P. Next, a path from an unmarked v_I on P to an unvisited v_T is traced out, then mark this v_I and v_T. With the traced graph, find another unmarked v_I and trace out another path. This procedure continues until no unmarked v_I can be found. Each time a path is traced out, an existing domain is divided into two subdomains, and a v_T and a v_I are marked. Since the number of v_T equals the number of components, there can be at most N − 2 paths after P. This in turn constrains the number of v_I to N − 2. □

In order to reduce the traversal time, all the edge-lists are collapsed to a succinct representation. Each edge-list is replaced by an edge, called a *simple edge*. After the transformation, the mating graph in Fig. 2.2(*b*) becomes a graph consisting only of v_I − v_I and v_T − v_I pairs, as shown in Fig. 2.3. If the span of the normals of an edge-list is not semi-circular, the simple edge is marked as non-monotonic; otherwise, the two extreme unit normals are recorded on the simple edge.

Lemma 2.4 The maximum possible number of simple edges in a simplified mating graph is (2N − 3).

[*Proof*] Since the number of v_I'*s* in a mating graph is at most N − 2, the maximum possible number of v_I − v_I pairs is N − 3. From Lemma 2.2, the number of v_I − v_T pairs is N. Thus, the maximum possible number of simple edges is 2N − 3. □

One v_T is chosen as the root to traverse the mating tree by a depth-first search procedure. During the traversal, the span of unit normals is updated whenever an unvisited simple edge is reached. Each update takes constant time because there are only four extreme unit normals to be tested each time, two of them are from the parent simple edge and the others from the child simple edge. If the span exceeds a semi-circle, the simple edge is marked as non-monotonic, and the descendants of the simple edge will not be searched further. Therefore, if a v_I − v_T edge is reached and it is also recorded as monotonic after updating the span, the result is a monotone chain from the root to the v_T. Such a search procedure is performed for every v_T, and the procedure is referred to as *Depth-First and Semi-Circle search* (DFSC).

Lemma 2.5 The time complexity for finding all monotone chains in a mating tree is Max {$O(N^2)$, $O(E)$} where E is the total number of mating edge.

[*Proof*] The procedure given above is equivalent to an exhaustive search that finds all the monotone chains in a mating tree. Before the DFSC search, it takes $O(E)$ time to compute the extreme unit normals for all edge-lists and collapse them. From Lemma 2.4, the number of simple edges is bounded by the number of components and it takes constant time to update the span of the extreme unit normals when an unvisited simple edge is reached, so the DFSC search requires $O(N)$ time for a fixed root, and thus $O(N^2)$ time for N v_T'*s*. Therefore, the procedure given above takes Max {$O(N^2)$, $O(E)$} time. □

For each monotone chain, the structure is cut into two subassemblies according to the mating chain and their separability is tested using the *separability algorithm* developed by Sack

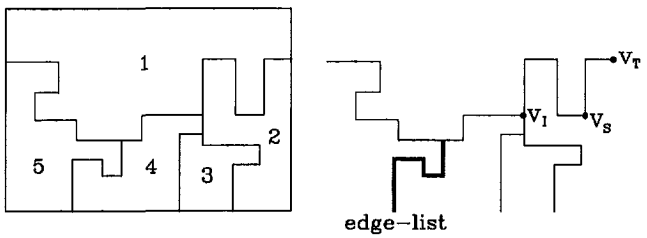


Fig. 2.2 (*a*) A 2D assembly (*b*) the mating graph of the assembly

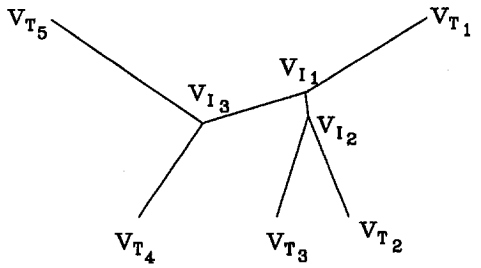


Fig. 2.3 Simplification of a mating graph

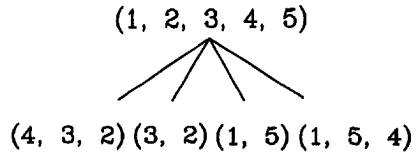


Fig. 2.4 The RT of Fig. 2.2(*a*)

and Toussaint (1987). The generation of a RT (Removability Tree) for the structure in Fig. 2.2(*a*) is shown in Fig. 2.4.

The *main-subassembly* is defined as the assembly remaining after the qualified components are removed. In order to distinguish the subassemblies in a RT, the RT nodes, except the root, are referred to as *node-subassemblies.* Because the node-subassemblies and the main-subassemblies are not tested further, the resulting RT has only two levels, including the root. This RT is referred to as a *partial RT.*

Cyclic Mating Graph. If there exists some components that are totally inside of an assembly, the mating graph of the assembly will contain cycles. Unlike the acyclic graph, a cyclic mating graph will have more than one route from one v_T to another v_T. In the example of Fig. 2.5, there are four routes from v_{T_1} to v_{T_2}.

Lemma 2.6 If the mating graph is cyclic, the time complexity for finding a monotone chain by traversing all v_T's is Max $\{O(2^N N^2), O(E)\}$.

[*Proof*] Since the boundary of each component may be considered as a cycle, for an assembly with N components, there will be $O(N)$ cycles in the mating graph. So there will be $O(2^N)$ routes between any pair of v_T's. Following the procedure described in Lemma 2.5, each fixed root will have $O(2^N N)$ mating chains here. Since there are $O(N)$ terminal vertices to be root in turn, the total time complexity for finding a monotone chain in a cyclic graph is therefore Max $\{O(2^N N^2), O(E)\}$. □

The Complete Solution in An Acyclic Mating Graph. By Lemma 2.5 and Lemma 2.6 a monotone mating chain can be computed by traversing a mating graph from each v_T. If the resulting main-subassembly or node-subassembly are disassembled further, the RT will be more than two levels for both graphs. For example, the complete solution for the structure in Figure 2.2(*a*) is shown in Fig. 2.6.

Lemma 2.7 The time complexity for constructing the complete RT in an acyclic mating graph is Max $\{O(N^{2N}), O(E)\}$.

[*Proof*] This is done by a recursive procedure. As a subassembly is disassembled, it will have many sub-subassemblies, and so on. Because every component contacts with the outside, from Lemma 2.2, the number of v_T's is equal to the number of components, so the time complexity is formulated as follows:

$T(N)$ represents the time complexity for finding the complete RT for a structure having N components. For the base case, where $N = 2$, the solution can be found in constant time, i.e., $T(2) = k$, k being a constant.

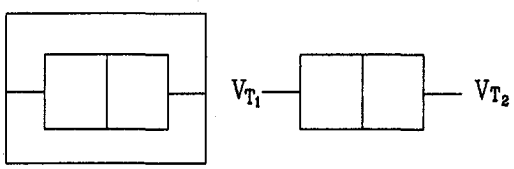


Fig. 2.5 An assembly with cyclic mating graph

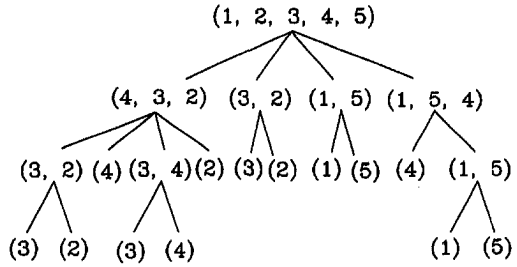


Fig. 2.6 Complete RT for Fig. 2.2(*a*)

If there are N components, then

$$\begin{aligned} T(N) &= N\{T(2) + T(3) + T(4) + \ldots + T(N-1)\} \\ &= N\{T(2) + 3T(2) + 4[T(2) + T(3)] + \ldots \\ &\quad + (N-1)[T(2) + T(3) + T(4) + \ldots + T(N-2)]\} \\ &= N\{O(N^2)[T(2) + T(3) + T(4) + \ldots + T(N-2)]\} \\ &= O(N^{2N}) \quad \square \end{aligned}$$

It is difficult to find all solutions if the structure is complicated and large. Instead of computing the complete solution, one practical approach is to generate an effective partial solution to reduce the time complexity. The strategy is to disassemble the node-subassembly that contains the query component and has the smallest number of components. The algorithm is illustrated as follows:

Algorithm 1: Remove_A_Component
Input: Assembly A and query component c
Output: RT1 (removability tree for reaching component c)
BEGIN
(1) Find the partial RT of assembly A;
(2) $S \leftarrow$ the smallest node-subassembly in the partial RT which contains component c;
(3) **If** $(S = A)$ **then**
(4) STOP;/* there is no feasible solution */
(5) **Else**
(6) **While** $(S - c \neq \emptyset)$
(7) **Algorithm 1** $\leftarrow S$;
END

Lemma 2.8 If only the smallest node-subassembly that contains the query component is disassembled further, it takes Max $\{O(N^3), O(E)\}$ time to construct a RT for reaching a query component when the mating graph is acyclic.

[*Proof*] By Lemma 2.5, it takes Max $\{O(N^2), O(E)\}$ to traverse a mating tree. After the partial RT is obtained, only the smallest node-subassembly which contains the query component is tested, and the RT will have N levels in the worse case. It takes $O(N^2) \times O(N) = O(N^3)$ to traverse all mating trees. Thus the time complexity is Max $\{O(N^3), O(E)\}$. □

Algorithm 1 finds a shortest path in the RT to reach a query component, which yields a sequence of disassembly to eventually disassemble the query component. The idea is to avoid removing components not affecting the disassembly of the query component. In other words, this greedy-typed algorithm tries to find a simplest way, simplest in terms of the number of components having to be disassembled, to remove a query component.

The same strategy can be employed to construct a RT to reach a particular component in a cyclic mating graph. In this case the time complexity is Max $\{O(2^N N^3), O(E)\}$. It is still very time consuming. The following section introduces an algorithm which reduces the time complexity when the mating graph is cyclic by taking the advantage of acyclic mating graphic.

3 Parallel Disassembly Algorithm

From Lemma 2.8, if every component contacts with the outside, it takes polynomial time to reach a particular component. But if there are some components totally inside a structure, it may take as much as exponential time to find a disassembly sequence to reach the query component.

We present an algorithm, using the attributes of the acyclic mating graph, which takes Max $\{O(N^3), O(E)\}$ time to find a heuristic solution to reach a particular component when the mating graph is cyclic.

In the algorithm implementation the winged-edge data structure (Baumgart 1972) is used to represent the relationship of the simple edges, components and vertices in a mating graph.

Outer Component Traversing. Recall that there are $O(2^N)$ paths connecting any two v_T's in a cyclic mating graph. Dutta and Woo (1992) developed a parallel disassembly procedure to address the special case in which no individual boundary component is removable. In that procedure, interior components are merged with boundary components if their adjacent edges are monotonic so that a removable subassembly may be determined. Although this monotonicity test for merging provides a necessary condition for removability, it is not a sufficient condition. Consider, for example, the assembly shown in Fig. 3.1, after determining that no single boundary component is removable, the Dutta and Woo (1992) algorithm searches for an interior component that is adjacent to a boundary component. Since the edge-list shared by component *2* and *4* is monotonic, they may be merged. However, the edge-list shared by component *3* and *4* is also monotonic, but the merged component (*3* and *4*) is not removable. Also, there may exist internal components which are not adjacent to any boundary component.

A different approach is taken in this research. Here, only the outer components are considered and *all* inner components are merged into one large loop called the *inner-loop,* regardless of the removability of any boundary component (See Fig. 3.2(*b*)). The resulting graph is called the *outer-graph.* If any inner component remains in the assembly after outer components are removed, it is assumed that these inner components can be merged to form only one inner-loop, and the process is repeated. By merging all interior components, the resulting outer graph is simplified and there are only two paths between any two v_T's, one is obtained by traversing the graph counterclockwise, the other by clockwise traversal. To further reduce computation time, the algorithm traverses the outer-graph in the counterclockwise direction when the inner-loop is reached, so there is *only one* path between any two v_T's. Considering again the assembly of Fig. 3.1, the *outer component traversing* approach would disassemble component *1* and *3* in parallel, leaving component *2* and *4* as the main-subassembly.

Maximum Set Insertion. In Algorithm 1 a partial RT is obtained after each current v_T has been traversed, then the smallest set which contains the query component in the partial RT

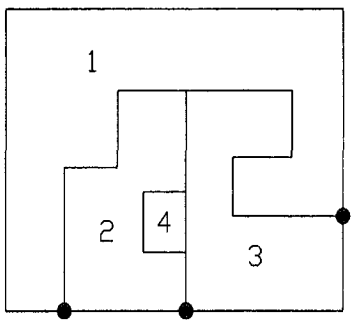


Fig. 3.1 Component 4 merged with component 3 creates an invalid assembly

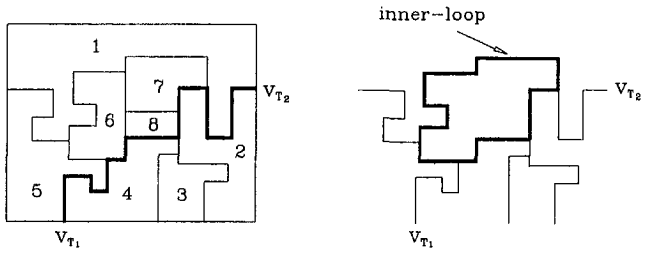


Fig. 3.2 (*a*) A 2D assembly (*b*) the inner-loop and outer-graph

is disassembled further. In a cyclic mating graph, however, if the query component is totally inside the structure, after the outer-graph has been traversed and all removable sets have been found, none of the sets would contain the query component.

Thus, in a cyclic mating graph, a *maximum set insertion* procedure is used to add the new nodes instead of all sets. In order to reach a particular component in a minimum number of steps, the maximum set insertion procedure generally sacrifices reaching the solution with the minimum number of components removed. In this paper, the goal is to find the shortest path in the RT to reach a particular component by the parallel disassembly algorithm and construct a unique parallel disassembly tree. After the outer-graph is traversed for each v_T, all sets whose intersections with the node-subassemblies in the current RT are not empty are deleted. Then, the largest set among the rest is chosen as the new node of the next level in the RT. Then, of the remaining sets, those whose intersections with the new node are not empty are deleted, and the next largest set among those remaining sets is inserted into the RT. The same procedure is repeated to insert other nodes into the RT until there are no sets left.

For example, in Fig. 3.3, suppose components *1, 2, 3, 4,* and *5* are the outer components of an assembly. $S_1\{1\}$, $S_2\{1, 2\}$, $S^3\{2, 3\}$, $S_4\{3, 4\}$, $S_5\{3, 4, 5\}$, and $S_6\{5\}$ are the removable combinations and no identical components exist in the current RT. Following the maximum set insertion algorithm only sets $S_5\{3, 4, 5\}$ and $S_2\{1, 2\}$ are inserted into RT. The reason for deleting the sets that have elements in common with the node-subassemblies in the current RT is to avoid the same component appearing in different nodes, which reduces the solution domain and time complexity. The reason for choosing the largest set is to reduce some trivial steps. For example, in Fig. 3.3, suppose the goal is to reach an inner component which is deeper than $\{1, 2, 3, 4, 5\}$, and components *3, 4 and 5* must be removed before it can be reached. $S_4\{3, 4\}$ can be removed first, then $S_6\{5\}$ is removed afterward. It takes two steps. If the largest set is chosen, only $S_5\{3, 4, 5\}$ is removed, and it takes only one step.

Outer-Graph Updating. After the current outer-graph has been traversed, the main-subassembly and corresponding outer-graph are updated. The renovation of the main-subassembly is affected by removing one leaf and all its ancestors from the original structure. The *original structure* means the complete structure without any component cleared. Thus, the original data structure is copied each time before the procedure, and the procedure is repeated until every leaf on the current level has been tested. After each renovation a new main-subassembly and

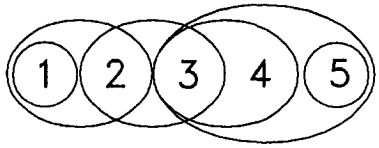


Fig. 3.3 The combination of disassemblable components

a new outer-graph are obtained. Then, the DFSC procedure is applied on the new outer graph and the maximum set insertion procedure is used to insert the new nodes.

Finally, a temporary disassembly tree is obtained. It is called the *basic-RT.* Since only outer components are considered during the traversal, the components in each node-subassembly are exposed to the outside and they appear only once in the basic-RT. Then, the basic-RT is searched to find the node-subassembly which contains the query component, which is then input into Algorithm 1 to find the final RT for the query component.

Parallel Disassembly Algorithm. The overall algorithm for finding a sequence to reach a particular component in an assembly is given by Algorithm 2: *Parallel_Disassembly_Algorithm,* which contains three procedures, namely, *Initialize, Traverse,* and Algorithm 1.

Before executing Algorithm 2, the procedure *Preprocessing* computes the unit normals for all mating edges.

Preprocessing
BEGIN
(1) Find the two extreme unit normals for each edge-list;
(2) Collapse all edge-lists to simple edges and record the corresponding two extreme unit normals on the simple edges;
END

Algorithm 2: Parallel_Disassembly_Algorithm
Input: Assembly A, and query component c
Output: RT
BEGIN
(1) **Initialize**();
(2) **Traverse**(A);
(3) $S_N \leftarrow$ Scan the current removability tree (basic-RT), and find the node-subassembly which contains c;
(4) **Algorithm 1** $\leftarrow S_N$;
END

The procedure *Initialize* initializes a basic-RT. The original structure is the root of the tree. *Initialize* also declares an array, *Temporary_Tank*[N][N], which stores the components in the removable sets.

Initialize()
BEGIN
(1) $lv \leftarrow 0$; /* level of the RT */
(2) Insert A as the node at level lv in RT;
(3) Declare an array, *Temporary_Tank*[N][N];
(4) Mark the root; /* the node in level 0 */
END

The procedure *Traverse* traverses the outer-graph of the input assembly by DFSC procedure. If any removable set is found during the traversal, it is stored in *Temporary_Tank.* If the current level is lv, use the *maximum set insertion* method to insert the qualified sets into level (lv + 1).

Traverse(assembly S)
BEGIN
(1) Initialize a circular linked list, *Cirlist;*
(2) Traverse the boundary of S in counterclockwise order to record the terminal vertices and outer components into *Cirlist;*
(3) $G \leftarrow$ outer-graph of S; /* All of the edges in G have been replaced by the simple edges in the preprocessing procedure */
(4) *setnum* $\leftarrow 0$;
(5) **For** (all v_T in *Cirlist*) **do** {
(6) $v_{begin} \leftarrow v_T$;
(7) **Take** v_{begin} as a root to traverse G by **DFSC** {
(8) **If** (a $v_I - v_T$ edge is reached and it is recorded as monotonic after updating the span) **then** {
(9) $v_{end} \leftarrow$ the v_T in the $v_I - v_T$ pair;
(10) Merge the components recorded from v_{begin} to $v_{end\text{-}1}$ in *Cirlist* as component P;
(11) Merge the components ($S - P$) as component Q;
(12) **Separability-Algorithm** $\leftarrow P$ and Q;
(13) **If** (P is separable) **then** {
(14) Store the components, contained in P, into *Temporary_Tank*[*setnum*];
(15) Increment *setnum* by one;}
}
}
}
(16) Delete the sets, stored in *Temporary_Tank,* whose intersections with the node-subassemblies are not empty;
(17) Use *maximum set insertion* method to insert nodes into level (lv + 1) $\leftarrow$ the sets remaining in the *Temporary_Tank;*
(18) **Level_Increase**();
END

If all the sets in level lv have been tested, the level of the tree will be increased by one; otherwise take one untested node and remove it and its ancestors from the original structure, then test the resulting assembly by the procedure *Traverse.* After the level is increased by one, if no set can be inserted, then the basic-RT has been constructed completely. The procedure *Level_Increase* implements the algorithm.

Level_Increase()
BEGIN
(1) Empty *Temporary_Tank;*
(2) **If** (there is unmarked node k in level lv) **then** {
(3) Mark node k;
(4) $S_N \leftarrow$ make one copy of the original data structure;
(5) $S_M \leftarrow$ main-subassembly updating; /*remove node k and its ancestors from S_N */
(6) **Traverse**(S_M);
}
(7) **Else** {
(8) $lv = lv + 1$;
(9) **If** (there is nonempty set in level lv) **then**
(10) **Level_Increase**();
(11) **Else**
(12) Stop;
}
END

In step 10 of the procedure *Traverse,* the components, recorded from v_{begin} to $v_{end\text{-}1}$ in *Cirlist,* are merged. This procedure is more robust than Algorithm SPLIT, proposed by Dutta and Woo (1992). Algorithm SPLIT disassembles components having faces in the mating chain. For example, in Fig. 3.2(a) component *4, 3,* and *2* can be removed according to the terminal vertices stored in *Cirlist* after the mating chain from v_{T1} to v_{T2} is found to be monotonic. In contrast, if Algorithm SPLIT is used, component *3* will be missed since none of its faces are on the mating chain.

Lemma 3.1 The time complexity for *Parallel_Disassembly_Algorithm* is Max $\{O(N^3), O(E)\}$.

[*Proof*] Because there are N components in the structure, the upper bound for the number of components which contact the

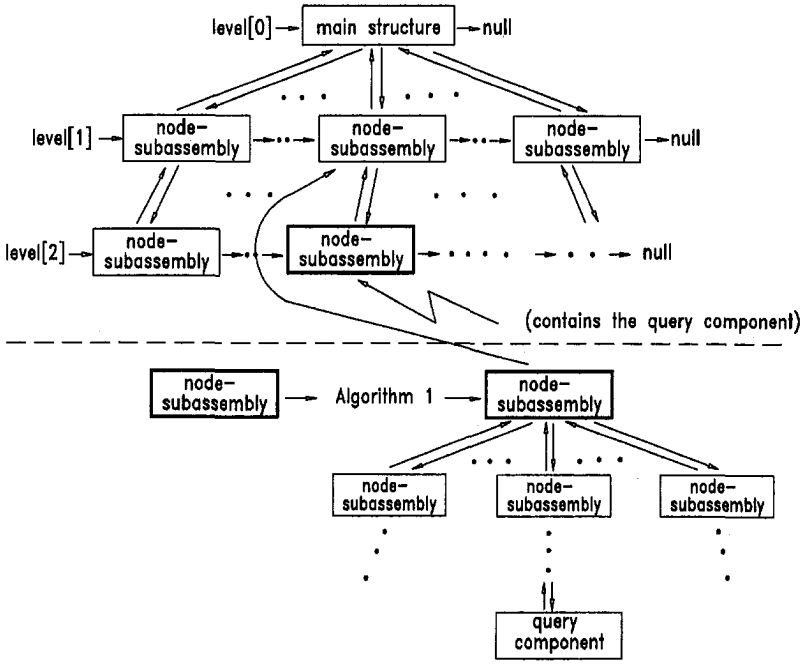


Fig. 4.1 The data structure of RT

outside is $O(N)$ for each layer, so it takes $O(N^2)$ to traverse the outer-graph. Since the number of layers is bounded by N, it takes $O(N^3)$ to construct a basic-RT. After that, the basic-RT is searched to find the node-subassembly which contains the query component and it is disassembled by Algorithm 1. Since it takes $O(E)$ for the preprocessing, the total time complexity is Max $\{O(N^3), O(E)\}$. □

4 Implementation

The parallel disassembly algorithm is implemented in the *C* language using the Graphic Library from Silicon Graphics Inc. Given input geometry and assembly files, and the query component, it will generate a disassembly sequence and show the procedure in graphical form.

The data structure which represents the final RT is described in Fig. 4.1. The RT above the dotted line is the basic-RT, which is generated by Step 1 to Step 3 of *Parallel_Disassembly_Algorithm*, and the RT below the dotted line is generated by Step 4. The basic-RT is searched to find the node which contains the query component, then the node-subassembly is disassembled by Algorithm 1.

Examples 1. Consider the assembly shown in Fig. 4.2. The parallel disassembly program operation is indicated as follows:

Input the edge data file: edge.inp
Input the component data file: comp.inp
Input the edge-list data file: edge-list.inp
Input the intersection and terminal vertices data file: vert.inp
1 2 3 4 5 6 7 8 9 10 11 12 13 14 → *(2 3 4)*
1 5 6 7 8 9 10 11 12 13 14 → *(1 5 6)(8 9 10)*
7 8 9 10 11 12 13 14 → none
1 5 6 7 11 12 13 14 → *(7 11 12 13 14)*
Which component do you want to move? *8*
8 9 10 → *(8 9)(8)(9 10)(10)*
8 → none
*(8)*moving direction is (1.000000, 0.000000)
*(8 9 10)*moving direction is (0.000000, −1.000000)
*(2 3 4)*moving direction is (0.000000, −1.000000) □

Before it prompts for the query component, the program computes the basic-RT as shown in Fig. 4.3. Because the query component is *8*, node *(8 9 10)* is disassembled by Algorithm 1,

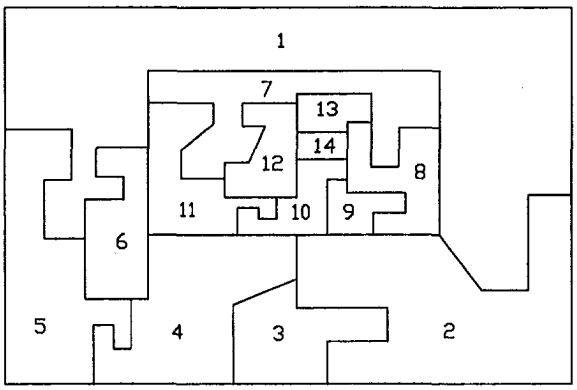


Fig. 4.2 Example structure

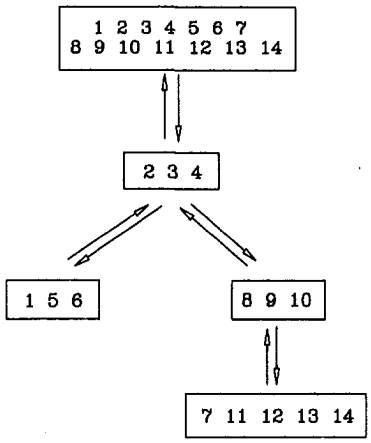


Fig. 4.3 The basic-RT of Fig. 4.2

so a disassembly sequence for component *8* is obtained: (*2 3 4*) → (*8 9 10*) → (*8*).

Examples 2. Consider the same assembly as example 1, but component *12* is requested.

Input the edge data file: edge.inp
Input the component data file: comp.inp
Input the edge-list data file: edge-list.inp
Input the intersection and terminal vertices data file: vert.inp
1 2 3 4 5 6 7 8 9 10 11 12 13 14 → (*2 3 4*)
1 5 6 7 8 9 10 11 12 13 14 → (*1 5 6*)(*8 9 10*)
7 8 9 10 11 12 13 14 → none
1 5 6 7 11 12 13 14 → (*7 11 12 13 14*)
Which component do you want to move? *12*
7 11 12 13 14 → (*11*)(*7 12 13 14*)(*14*)(*7 11 12 13*)(*13 14*)(*7 11 12*)
7 11 12 → (*11*)(*7 12*)
7 12 → none
(*7 12*)moving direction is (1.000000, 0.000000)
(*7 11 12*)moving direction is (0.000000, 1.000000)
(*7 11 12 13 14*)moving direction is (0.000000, −1.000000)
(*8 9 10*)moving direction is (0.000000, −1.000000)
(*2 3 4*)moving direction is (0.000000, −1.000000) □

When the main-subassembly just contains components *7* and *12,* component *12* is disassemblable, but it is inseparable by 1-translation motion, so it is still not removable.

4 Conclusion

The difficulty of the disassembly problem is the inherent complexity of possible solutions. This paper develops an algorithm to find a heuristic disassembly sequence to remove a component out of a structure by parallel disassembly procedures. Since it is easier to analyze the structure that does not contain internal components, the structure is disassembled into several modules, which do not contain internal components. The basic-RT is constructed first with each component appearing only once in the node-subassemblies. The basic-RT is identical for every query component, and the node-subassembly to be considered is determined based on the input query component. The time complexity of the algorithm is Max $\{O(N^3), O(E)\}$, where N is the number of components and E is the total number of mating edges.

Acknowledgment

The authors are grateful for research support from the Office of Naval Research (Grant No. N0001492J4092) and the general support of the Iowa Center for Emerging Manufacturing Technology. The authors also sincerely appreciate helpful comments from the anonymous reviewers.

References

Baumgart, B. G., 1972, "Winged Edge Polyhedron Representation," Stanford University Department of Computer Science, Stanford Technical Report No. CS-72-320.

De Floriani, L., and Nagy, G., 1989, "A Graph Model for Face-to-Face Assembly," *Proc. of IEEE Conference on Robotics and Automation,* Vol. 1, pp. 75–78.

Dehne, F., and Sack, J. R., 1987, "Translation Separability of Sets of Polygons," *The Visual Computer,* Vol. 3, pp. 227–235.

Dutta, D., and Woo, Anthony C., 1992, "Algorithms for Multiple Disassembly and Parallel Assemblies," ASME *Concurrent Engineering,* PED-Vol. 59, pp. 257–266.

Homen de Mello, L. S., and Sanderson, A. C., 1990, "And/Or Graph Representation of Assembly Plans," *IEEE Transactions on Robotics and Automation,* Vol. 6, No. 2, pp. 188–199.

Lee, S., and Shin, Y. G., 1990, "Assembly Planning Based on Geometric Reasoning," *Computers and Graphics,* Vol. 14, No. 2, pp. 237–250.

Lee, S., and Wang, F. C., 1993, "Physical Reasoning of Interconnection Forces for Efficient Assembly Planning," *Proc. of IEEE Conference on Robotics and Automation,* Vol. 2, pp. 307–313.

Sack, J. R., and Toussaint, G. T., 1987, "Separability of Pairs of Polygons Through Single Translations," *Robotica,* Vol. 5, pp. 55–63.

Toussaint, G. T., 1989, "On Separating Two Single Polygons by a Single Translation," *Discrete and Computational Geometry,* Vol. 4, No. 3, pp. 265–278.

Woo, T. C., and Dutta, D., 1991, "Automatic Disassembly and Total Ordering in Three Dimensions," ASME *Journal of Engineering for Industry,* Vol. 113, pp. 207–213.